First we will upload the dataset

```
from google.colab import files
uploaded = files.upload()
```

Choose Files heart.csv

- **heart.csv**(text/csv) - 35921 bytes, last modified: 4/26/2025 - 100% done

```
Saving heart.csv to heart (1).csv
```

Now lets read the dataset

```
import pandas as pd
df = pd.read_csv('heart.csv')
print(df.head())
```

```
   Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  MaxHR  \
0   40   M           ATA        140          289          0     Normal    172
1   49   F           NAP        160          180          0     Normal    156
2   37   M           ATA        130          283          0         ST     98
3   48   F           ASY        138          214          0     Normal    108
4   54   M           NAP        150          195          0     Normal    122

  ExerciseAngina  Oldpeak ST_Slope  HeartDisease
0              N      0.0       Up             0
1              N      1.0     Flat             1
2              N      0.0       Up             0
3              Y      1.5     Flat             1
4              N      0.0       Up             0
```

Now we will check for missing value

```
print("\nChecking for missing value...")
print(df.isnull().sum())
```

```

Checking for missing value...
Age               0
Sex               0
ChestPainType     0
RestingBP         0
Cholesterol       0
FastingBS         0
RestingECG        0
MaxHR             0
ExerciseAngina    0
Oldpeak           0
ST_Slope          0
HeartDisease      0
dtype: int64
```

No missing data. So moving on to next step. Displaying input and output features of dataset.

```
print("\nInput features:")
print(df.columns[:-1].tolist())

print("\nOutput feature:")
print(df.columns[-1])
```

```

Input features:
['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS', 'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope']

Output feature:
HeartDisease
```

Now encoding non-numeric input attributes using Label Encoder.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for col in df.columns:
  if df[col].dtype == "object":
```

```
    df[col] = le.fit_transform(df[col])

print("labeled\n", df.head())
```

```
labeled
    Age  Sex  ChestPainType  RestingBP  Cholesterol  FastingBS  RestingECG  \
0   40    1              1        140          289          0           1
1   49    0              2        160          180          0           1
2   37    1              1        130          283          0           2
3   48    0              0        138          214          0           1
4   54    1              2        150          195          0           1

   MaxHR  ExerciseAngina  Oldpeak  ST_Slope  HeartDisease
0    172               0      0.0         2             0
1    156               0      1.0         1             1
2     98               0      0.0         2             0
3    108               1      1.5         1             1
4    122               0      0.0         2             0
```

Splitting Dataset into X and y

```
X = df.drop(columns=[df.columns[-1]]) #other column
y = df[df.columns[-1]] #target column
```

Normalizing using Standard Scaler

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

print(X)
```

```
     Age  Sex  ChestPainType  RestingBP  Cholesterol  FastingBS  RestingECG  \
0     40    1              1        140          289          0           1
1     49    0              2        160          180          0           1
2     37    1              1        130          283          0           2
3     48    0              0        138          214          0           1
4     54    1              2        150          195          0           1
..   ...  ...            ...        ...          ...        ...         ...
913   45    1              3        110          264          0           1
914   68    1              0        144          193          1           1
915   57    1              0        130          131          0           1
916   57    0              1        130          236          0           0
917   38    1              2        138          175          0           1

     MaxHR  ExerciseAngina  Oldpeak  ST_Slope
0      172               0      0.0         2
1      156               0      1.0         1
2       98               0      0.0         2
3      108               1      1.5         1
4      122               0      0.0         2
..     ...             ...      ...       ...
913    132               0      1.2         1
914    141               0      3.4         1
915    115               1      1.2         1
916    174               0      0.0         1
917    173               0      0.0         2

[918 rows x 11 columns]
```

Splitting Dataset into test set and train set

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

Now lets build MLP model 11x128x64x32x1 and train the model

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras import Input
model = Sequential([
    Input(shape=(11,)),
```

```
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.1, verbose=1)
```

```
Epoch 1/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 3s 18ms/step - accuracy: 0.6794 - loss: 0.6374 - val_accuracy: 0.8108 - val_loss: 0.4943
Epoch 2/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8889 - loss: 0.3888 - val_accuracy: 0.8108 - val_loss: 0.4221
Epoch 3/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8699 - loss: 0.3468 - val_accuracy: 0.7973 - val_loss: 0.4267
Epoch 4/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8834 - loss: 0.2992 - val_accuracy: 0.7973 - val_loss: 0.4525
Epoch 5/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8831 - loss: 0.2883 - val_accuracy: 0.8108 - val_loss: 0.4100
Epoch 6/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8913 - loss: 0.2550 - val_accuracy: 0.7973 - val_loss: 0.4145
Epoch 7/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8838 - loss: 0.2620 - val_accuracy: 0.7973 - val_loss: 0.4279
Epoch 8/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8929 - loss: 0.2402 - val_accuracy: 0.8108 - val_loss: 0.4416
Epoch 9/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8993 - loss: 0.2349 - val_accuracy: 0.8378 - val_loss: 0.4188
Epoch 10/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9051 - loss: 0.2377 - val_accuracy: 0.8108 - val_loss: 0.4747
Epoch 11/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9153 - loss: 0.2190 - val_accuracy: 0.8243 - val_loss: 0.4477
Epoch 12/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9193 - loss: 0.1998 - val_accuracy: 0.8108 - val_loss: 0.4958
Epoch 13/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9386 - loss: 0.1801 - val_accuracy: 0.8108 - val_loss: 0.5005
Epoch 14/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9240 - loss: 0.1964 - val_accuracy: 0.8243 - val_loss: 0.4898
Epoch 15/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9358 - loss: 0.1760 - val_accuracy: 0.8108 - val_loss: 0.4939
Epoch 16/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9419 - loss: 0.1728 - val_accuracy: 0.8108 - val_loss: 0.5365
Epoch 17/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9385 - loss: 0.1714 - val_accuracy: 0.7973 - val_loss: 0.5406
Epoch 18/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9312 - loss: 0.1962 - val_accuracy: 0.7973 - val_loss: 0.5939
Epoch 19/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9402 - loss: 0.1680 - val_accuracy: 0.7973 - val_loss: 0.6148
Epoch 20/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9529 - loss: 0.1573 - val_accuracy: 0.8108 - val_loss: 0.5602
Epoch 21/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9472 - loss: 0.1542 - val_accuracy: 0.8108 - val_loss: 0.6217
Epoch 22/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9625 - loss: 0.1376 - val_accuracy: 0.7973 - val_loss: 0.5664
Epoch 23/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9596 - loss: 0.1246 - val_accuracy: 0.7973 - val_loss: 0.6241
Epoch 24/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9646 - loss: 0.0999 - val_accuracy: 0.7838 - val_loss: 0.6670
Epoch 25/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9573 - loss: 0.1084 - val_accuracy: 0.7973 - val_loss: 0.6859
Epoch 26/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9668 - loss: 0.1001 - val_accuracy: 0.7973 - val_loss: 0.6810
Epoch 27/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9580 - loss: 0.1030 - val_accuracy: 0.7973 - val_loss: 0.6730
Epoch 28/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9624 - loss: 0.1044 - val_accuracy: 0.7838 - val_loss: 0.7827
Epoch 29/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9715 - loss: 0.0772 - val_accuracy: 0.7838 - val_loss: 0.7631
```

Now model should be able to predict. Displaying confusion matrix, accuracy, recall, precision and F1-score.

```
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score, precision_score, f1_score

y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int).flatten()


conf_matrix = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
```

```
f1 = f1_score(y_test, y_pred)

print("\n=== Model Evaluation ===")
print("Confusion Matrix:\n", conf_matrix)
print(f"Accuracy: {accuracy:.4f}")
print(f"Recall: {recall:.4f}")
print(f"Precision: {precision:.4f}")
print(f"F1-Score: {f1:.4f}")
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step

=== Model Evaluation ===
Confusion Matrix:
 [[65 12]
 [19 88]]
Accuracy: 0.8315
Recall: 0.8224
Precision: 0.8800
F1-Score: 0.8502
```